

Christopher Jones
October 24, 2016

CHEP 2016 Summary

Frameworks and Parallelization

Introduction

There were only a handful of talks about Frameworks at this year's CHEP. There were more talks about parallelization which covered the range from multi-threading, multi-processing, vectorization, GPUs and FPGAs.

Frameworks

ATLAS

ATLAS presented a talk about their recent work on their version of AthenaMT¹. They have concentrated on getting the framework components to work properly in a multi-threaded environment and will look into making the experiment algorithms thread-friendly in 2017.

They have been primarily concentrating on their non-event infrastructure Services in particular Conditions and Incidents. ATLAS Services are similar to CMSSW Services in that they are global accessible to all code. Unlike CMSSW Services, which are not allowed to affect physics results, ATLAS Services are used for all data passing between framework modules.

Conditions access is done through the ATLAS Conditions Service. Unlike CMS where we only update conditions on luminosity block boundaries, the ATLAS conditions can change on any arbitrary event boundary. Therefore ATLAS has worked to allow multiple instances of the same condition, which are for different intervals of validity or IOV, be accessible within the framework. They accomplish this via two changes. The first change is to require pre-registration of what conditions data each algorithm will be using. This pre-registration gives back a ConditionsHandle which is then used to retrieve the data. The second part is the Conditions service was changed to hold containers of conditions object where each instance is for a different IOV. Talking with the developers afterwards they say they have not yet figured out how to 'garbage collection' conditions for IOVs that are no longer needed. As part of this change, they have now moved their geometry to be managed by the Conditions Service.

Incidents in Athena are the way it handles non-event state changes. Formerly, incidents could be fired by any piece of code and reacted to by any other piece of code. This was problematic for thread-safety. Upon further review, they determine that the system was far more flexible than they needed so now incidents are transitions known and scheduled by the framework. This matches how CMSSW handles such transitions.

¹ <https://indico.cern.ch/event/505613/contributions/2230829/attachments/1347615/2039402/Oral-191.pdf>

The last bit of thread-safety work done to AthenaMT was to allow ‘re-entrant’ modules which are inspired by the CMSSW global modules.

A non-thread related recent change to Athena was the ability to support EventViews. An Event-View works like an EventStore exception it only allows access to information pertaining to a particular ‘region of interest’. This is intended to be used by their high level trigger system.

Future Circular Collider

The Future Circular Collider (FCC) design studies want to share software across the various studies as well as to use software already in use by other experiments². They are using Gaudi for some of their work. However, they are using PODIO³ as their data model. The reason they state is it was built with threading in mind (which as far as I can tell is it follows the same rules as CMS data products) and treats python and C++ on equal footing. They are also using the Heppy framework for their analysis (and code development?) framework. They specifically state that share Heppy with CMS.

LHCb

LHCb will be extending Gaudi using ideas from the GaudiHive project, in particular, the task scheduling system.⁴ They mention changing algorithms to be re-entrant (based again on CMS’ global modules) and that their data products must become immutable (which is something enforced in CMSSW).

On the event data model front, they want to be able to make better use of CPU vectorization. To that end they are making their data products read only, not use inheritance, use single precision where possible and allow different representations (e.g. ‘array of structures’ or ‘structures of arrays’). They are investigating the possible use of PODIO.

FairMQ

Compressed Baryonic Matter (CBM) and PANDA are exploring the possibility of using FairMQ framework for their online (and offline?) data processing.^{5,6} Both experiments use trigger less readout at a very high rate and will do data selection strictly in software. Both experiments therefore do not deal with individual events but instead deal with time slices. A time slice can contain multiple events and one event can span more than one time slice. FairMQ is a multi-process based framework which uses inter-process message queuing to pass data from one component to the next. The framework provides configuration, process management and monitoring tools. The processes controlled by FairMQ can reside on the same system or different

² <https://indico.cern.ch/event/505613/contributions/2230842/attachments/1347109/2036951/Oral-322.pdf>

³ <http://cds.cern.ch/record/2212785/files/AIDA-2020-NOTE-2016-004.pdf?version=1>

⁴ <https://indico.cern.ch/event/505613/contributions/2241722/attachments/1347463/2041645/Bozzi-CHEP2016.pdf>

⁵ <https://indico.cern.ch/event/505613/contributions/2241723/attachments/1349201/2042791/Oral-410-v2.pdf>

⁶ <https://indico.cern.ch/event/505613/contributions/2227258/attachments/1347539/2043446/Oral-104.pdf>

systems. In addition the processes can be written in different languages (C, C++ and FORTRAN were given as examples) and the processes themselves could be run on different operating systems. For FairMQ, task parallelization is accomplished by creating a series of different processes, each doing specific tasks, and having the message queues of these processes connected to one another to form an application chain. Timeslice (or event) parallelization is accomplished by making separate replicas of one application chain for each Timeslice to be processed concurrently. There was no mention about how this framework might be used within the context of a standard grid style batch system, although CBM did say they are thinking about only storing RAW data and having all users do reconstruction on the fly.

ALICE

The ALICE collaboration built a mini framework to study the effectiveness of different algorithms on KNC/KNL hardware.⁷ This framework isolates the algorithm from the host. Their approach is to have independent threads each doing one particular task and then connect these threads with thread-safe FIFOs. This matches their use of FairMQ. Their design is to have one thread on the host system which reads data from a FIFO on the local host, fills entries on a FIFO on the KNC/KNL system. The same host thread pulls data off a FIFO on the co-processor and puts the data out on a FIFO on the host. This requires only one host thread to have to interact with the KNC/KNL system. On the KNC/KNL system there is one thread handling the input from the host and then filling a FIFO feeding into the algorithm being tested. Then another thread reading the FIFO which was filled by the algorithm and writes to the FIFO read by the host. They showed some simple benchmarks using bzip as the test algorithm where they had multiple threads each running bzip independently. They ran tests where the algorithm was run on both a KNC and a KNL system and then compared to the same algorithm run on a Sandybridge 8 core system with two hyper-threads. They showed the KNC system ran only 1/5th as fast as the Sandybridge system and the KNL was only 3/4th as fast as the Sandybridge.

Parallelization

ROOT

ROOT 6.08 is adding additional parallelization options to the users.⁸ They have a multi-process (ROOT::TProcessExecutor) and multi-threaded (TProcessExecutor and ROOT::TThreadExecutor) classes which allow map/reduce style parallelism. The multi-threaded code is built on TBB although they do not explicitly expose the TBB interface to the user. They also can use threads implicitly (if directed) when reading data from a TTree. They are also providing a few thread helper items such the ROOT::TThreadedObject<T> class which provides a copy of an object for each thread and then a facility to merge all the copies into one final object.

In addition to what is now available, the ROOT team is working on two R&D projects. The first is exploring if it is possible to use functional programming concepts when expressing analysis. The second is to integrate Spark and ROOT using PyROOT and 'just-in-time' compiled C++. They showed some very preliminary results from such an integration.

⁷ <https://indico.cern.ch/event/505613/contributions/2228467/attachments/1345422/2041671/Oral-v3-349.pdf>

⁸ https://indico.cern.ch/event/505613/contributions/2228338/attachments/1346729/2039459/ParallelismInROOT_v4.pdf

Offline Tracking

LHCb is exploiting vectorization in their tracking reconstruction code.⁹ During track finding, they use SSE to calculate the Hough projection of two hits in parallel. They find that even after filling the SSE vector they gain a 40% speed improvement. During track fitting they used SIMD for transportation of covariance matrix from state. They gained a factor of 2 speedup by explicitly writing out the calculation and another 5x by using explicit AVX instructions.

CMS has a project which is exploring Kalman filter tracking on parallel architectures.¹⁰ They are working on exploiting both vectorization and parallelization. For now they are using a simplified detector description as they work out the design. They are using Kalman filter for both track fitting and track finding. When looking at vectorization performance, they see a factor of 4x when doing track fitting but only 2x for track finding. The latter lower result comes from the fact that track finding requires the code to branch often. As for parallelization, the developers switch from using OpenMP to TBB and were able to obtain much better scaling. On an Intel KNC co-processor they can get near perfect scaling up to 61 threads for both track fitting and finding and reach maximum of 100x speedup for track fitting and 85x for track finding.

A non-experiment specific talk was given by a project studying the application of Hough transform to exploit parallelism.¹¹ They chose to use the Duda-Haart-Hough Transform which uses polar coordinates for the transformation which converts a point into a sinusoid. The intersection of sinusoids defines a line in the untransformed space. To decrease the computational complexity, they discretized the transformed space into a NxN matrix and then look for high accumulations in the bins of the matrix. Filling the matrix and looking for 'hot spots' can be done in parallel. They showed that their implementation was more robust against noise than the standard Hough-transform method. Although their algorithm scaled well as a function of number of threads, it slowed down dramatically as the size of the matrix increased: 1000x slower when the number of matrix points grew by 64x. The developers plan to further their study.

Detector Simulation

There were two talks about running Geant 4 on Intel many-core (KNC/KNL) architectures. The first was from the Geant 4 team.¹² They showed that they have extended the Geant 4 to allow MPI to be used on-top of their existing threading model. They have continued improving their multi-threaded application by decreasing the per thread memory requirement by more than factor of 2. When run on the KNL system, they find >93% efficiency when they use as many threads as cores in their different test applications. They were able to get a 50% better event throughput using the KNL system over a 12 core x 2 hyper threaded Xeon processor. They have also begun testing the MPI based system on super-computers (Mira@ANL). They scale well to 64k threads but then hit a limit which they attribute to I/O. The second talk was from the ATLAS

⁹ <https://indico.cern.ch/event/505613/contributions/2230861/attachments/1347027/2038311/Oral-567.pdf>

¹⁰ <https://indico.cern.ch/event/505613/contributions/2254599/attachments/1347645/2043445/Oral-115-v8.pdf>

¹¹ <https://indico.cern.ch/event/505613/contributions/2228349/attachments/1346689/2044379/Oral-436.pdf>

¹² <https://indico.cern.ch/event/505613/contributions/2228330/attachments/1343565/2043243/Oral-161.pdf>

team.¹³ They are looking towards using the Xeon Phi based super-computers such as Cori@NERSC. They believe simulation would be a good fit to super-computers since it is very CPU intensive and has little input needs. However, they are concerned that memory usage and lack of vectorization may prohibit efficient utilization of the system. In their test, they were able to allow concurrent processing of all available parts of their simulation except for the handling of output. When running on the KNL, they see excellent scaling up to the number of physical cores and substantial gains when using hyper-threads. When using all the available threads, the 1.3GHz Xeon Phi had event throughput equivalent to 16x2 2.6GHz Xeon processor (which was about 0.045 events/second). When processing a much simpler generator sample of just one muon, they found that above 180 threads (or 23 events/second) their throughput actually substantially decreased. The decrease was due to the output hitting its single-threaded limit. To better understand the performance characteristics of the simulation code running on the KNL machine, they profiled the code using Intel's VTune. What they found was a very high instruction cache miss rate.

Fermilab presented results on GeantV EM Physics Models.¹⁴ The work was done to explore the feasibility of leveraging vectorization for physics model calculations using either vector units of CPUs or GPUs. EM Physics was chosen since it takes approximately 30% of the total simulation time (for a standalone CMS based benchmark). Since the standard sampling algorithm used by Geant4 for modeling the physics involves many branches, an alternative algorithm (the Alias Method) was used. The results show a speed up on Intel KNL type architectures of 3-6 when used on 10s of tracks while a gain of 30 for GPUs but that requires 10^4 tracks to be processed simultaneously. Once the full work is completed, the code will be integrated into the GeantV simulation system.

Another aspect of GeantV presented in a talk was VecGeom.¹⁵ VecGeom is a 3D geometric shape library and navigation system which is designed to exploit vectorization by either processing multiple tracks through the same volume or by doing multiple volume calculations (e.g. checking for distance to all sides) concurrently. They now have implemented all of the shape primitives needed by most HEP experiments. Using these new shapes within ALICE code gave speedups of up to 9x. They also showed that navigation (e.g. finding the next volume along a straight line) also benefits from vectorization. They implemented an algorithm which could calculate intersections for child volumes concurrently within a parent volume. They were able to demonstrate a greater than 2x speedup.

Generators

An effort was undertaken to have Sherpa run on supercomputers.¹⁶ This was done using MPI to parallelize the work done on the phase space integration task (which is the most computational-

¹³ <https://indico.cern.ch/event/505613/contributions/2230831/attachments/1347662/2047237/Oral-v5-196.pdf>

¹⁴ <https://indico.cern.ch/event/505613/contributions/2228329/attachments/1347600/2041612/Oral-158-v2.pdf>

¹⁵ https://indico.cern.ch/event/505613/contributions/2228346/attachments/1349199/2045049/Oral_393.pdf

¹⁶ <https://indico.cern.ch/event/505613/contributions/2228354/attachments/1347575/2043279/ORAL-v1-2228354-childers.pdf>

ly expensive task in Sherpa). The initial results showed little to know scaling as the number of nodes were increased. They tracked the scaling problem to the fact that all MPI ranks were required to finish computing 5 phase space points before synchronizing and starting the next phase. However, some phase space points take longer to calculate so the ranks had to wait for the slowest one to catch up. Changing the ranks to stop calculating after a fix time, rather than fix amount of work, allowed them to achieve good scaling on KNL based architectures.

Machine Learning

Machine learning was discussed several times with respect to parallelization. The TMVA developers discussed the work they have done to use CPU threads and GPUs when training a deep neural network.¹⁷ It appears using multiple threads give a factor of 5 speedup and a factor of 24 speedup both compared to the single threaded implementation. LHCb showed results using a neural network trained to find fake tracks in their HLT.¹⁸ It appears that they use hand written vectorization code to speedup the neural net calculation.

Online Systems

Given their strict control over the computing hardware used, the online groups are exploring the use of non CPU based parallel architectures, in particular GPGPUs and FPGAs.

ATLAS is exploring the use of GPGPUs in its high level trigger for Run 3.¹⁹ All of their tests make use of a separate server process that is independent from their framework and handles all interactions with the GPUs. This server process is shared by all the framework processes running on the machine. They have been testing GPU algorithms for track seeding (which is 5x faster), calorimeter cluster finding (1.3-2x faster) and muon track finding (no speedup given). In total they see a 20-40% event throughput gain and the server process can efficiently serve 14 framework processes.

ALICE has been using GPUs for tracking in their online system since Run 1.²⁰ Their present system is sufficient for Run 2 so they are concentrating on changes for Run 3. To improve their efficiency they want to exploit the new GPU hardware's ability to handle multiple kernels simultaneously. This allows them to utilize kernels in the case where there are not enough tracks in one geometric region to fill the entire GPU. Along this same line, they are also testing filling the GPU with kernels/data from multiple events. Although this increased the time to process one event, the average time for all events decreased by 30%. The developers next want to explore moving all the tracking steps to the GPUs to avoid copying to/from the host between the different steps.

¹⁷ <https://indico.cern.ch/event/505613/contributions/2228344/attachments/1347106/2041567/oral-CHEP16-SergeiVGleyzer.pdf>

¹⁸ <https://indico.cern.ch/event/505613/contributions/2230861/attachments/1347027/2038311/Oral-567.pdf>

¹⁹ <https://indico.cern.ch/event/505613/contributions/2227286/attachments/1348002/2046973/Oral-482.pdf>

²⁰ <https://indico.cern.ch/event/505613/contributions/2227277/attachments/1348662/2039067/paper-387.pdf>

CMS had two talks about exploring the use of GPUs in the HLT for future upgrades. One talk was about tracking²¹ and the second was about clustering in calorimetry²². The tracking project is working on passing raw data to the GPU and then gets back tracks. For this talk they only discussed track finding. To better utilize parallelization, they changed from using a 'triplet propagation' algorithm for track finding to one based on cellular automata (CA). They have implemented the algorithms on both the GPU and CPU and get the same results. When comparing the CA algorithm to the triplet algorithm they find greater track finding efficiency with an acceptable increase in fake rates. Looking at the time performance of the algorithms, the CPU version of the CA algorithm is 3x faster than the triplet and the CPU version of CA was 14x faster than its CPU version. The calorimetry based talk was discussing how to handle the sampling calorimeter which will be placed in the end caps of CMS. The problem is to find all the nearest neighbors for 300,000 points within a few milliseconds. They chose an algorithm based on FKDTTree which has the following GPU friendly features: no recursion, just iteration and limited branching. Testing found a speedup of 9 to 15 when comparing the GPU algorithm to a serial CPU version of the same algorithm.

LHCb has tested the use of a GPU tracking algorithm in their HLT.²³ They took their existing fast tracking algorithm and rewrote it in CUDA to run on the GPU. To get sufficient parallelization, they feed data from multiple events into the GPU algorithm. They were able to test this online in their HLT but sending data to a node containing a GPU. The node was running multiple instances of Gaudi and each of these instances ran tracking on the CPU as well as sent data to and the from the GPU to do equivalent tracking. This allowed them to compare the results of the CPU to the GPU. The Gaudi clients all sent data to an external process which handled communication with the GPU. This allowed them to queue up data from multiple events to be processed simultaneously by the GPU. They found the results to be statistically similar (but not identical) between the CPU and GPU algorithms and the GPU was slower than the CPU version of the algorithm by 30-80% depending on how many Gaudi clients were sharing the GPU (with more clients creating a better rate).

There were five talks about the use of FPGAs. One talk implemented a Convolutional Neural Network algorithm on the FPGA to process signals from Micromegas detectors.²⁴ They obtained reasonable results but were limited by the precision of the data they were able to feed into the FPGA. A second talk was from LHCb and their exploration of Intel Xeon/FPGA prototype applied to their RICH particle id algorithm.²⁵ They were able to get a 35x speedup but they were limited by the data transfer to the FPGA. They also stressed that using OpenCL to program the FPGA

²¹ https://indico.cern.ch/event/505613/contributions/2227276/attachments/1348461/2045263/Felice_CHEP.pdf

²² <https://indico.cern.ch/event/505613/contributions/2227279/attachments/1349993/2045264/chep-fkdttree.pdf>

²³ <https://indico.cern.ch/event/505613/contributions/2227265/attachments/1346639/2043332/Oral-238.pdf>

²⁴ https://indico.cern.ch/event/505613/contributions/2227234/attachments/1347570/2041446/Oral_v3_004.pdf

²⁵ <https://indico.cern.ch/event/505613/contributions/2227272/attachments/1346688/2045079/Oral-312.pdf>

was much better than the traditional approach. A third talk was from ALICE who uses FPGAs already in their HLT and is using those to test ideas for Run 3.²⁶ The FPGAs are presently used to readout the data from the detector and now they are exploring having the FPGAs run cluster finding reconstruction code directly on the data read by the hardware. A fourth talk outlined a track finding algorithm inspired by human vision and which was suitable for parallelization using FPGAs.²⁷ The device encodes all possible tracking patterns and then finds the best match. From the simulations they ran they believe the approach is promising for the HL-LHC. The fifth talk was from CMS and covered track finding in the level 1 trigger for the HL-LHC.²⁸ This would have to run at 40MHz with a latency of only 4 μ s. Two approaches are being tested. The first approach uses associated memories to do pattern recognition and then an FPGA to do track fitting. The second approach uses Hough transform algorithm on the FPGA to do the pattern recognition as well as the final track fitting. They are still working towards having a system demonstration by the end of 2016.

²⁶ <https://indico.cern.ch/event/505613/contributions/2227259/attachments/1346753/2030983/Oral-113.pdf>

²⁷ <https://indico.cern.ch/event/505613/contributions/2227295/attachments/1346720/2045223/Oral-v1-555.pdf>

²⁸ <https://indico.cern.ch/event/505613/contributions/2227281/attachments/1350066/2045082/Oral-404.pdf>